`form` code for the decay of the Higgs boson

Jean-Philippe Guillet

February 4, 2020

1 Code FORM

The purpose of this note is not to give a course on `form`, but only to present an example with some explanations and we refer the readers to the full reference:
`http://www.nikhef.nl/∼form/maindir/documentation/reference/online/`
It exists also some courses on the web.

A `form` program is made as a sequence of modules. A module consists in general of several types of statements (in the right order):

- Declarations: these are the declarations of variables.
- Specifications: these tell what to do with existing expressions as a whole.
- Definitions: these define new expressions.
- Executable statements: the operations on all active expressions.
- OutputSpecifications: these specify the output representation.
- End-of-module specifications: extra settings that are for this module only.

We will decompose the program `form` for the computation of the decay of the Higgs boson via W loops in term of the various modules.

Module 1

```
*
* Reaction H(q) --> Gamma(p1)+Gamma(p2) : W loops
*
Vector p1,p2,q,k,[k+p2],[k+p1],[k+q],l;
```

```
5 Symbol Mw,Mz,q2,x,y,[1/n],l2,R2,n,D1,D2,D0,D3;
6 * Uncomment this line
7 *Indice alpha1=4,alpha2=4,beta1=4,beta2=4,mu1=4,mu2=4;
8 * and comment the following line for a computation in four dimension
9 Indice alpha1=n,alpha2=n,beta1=n,beta2=n,mu1=n,mu2=n;
10 * end comment
11 *
12 * The differents diagrams are split in two parts
13 * R contains the common part : the vertex HWW and the two
14 * adjacent W's propagators
15 * M1 contains the two couplings GammaWW and the extra W propagator
16 * M2 : same thing as M1 with p1 <--> p2
17 * M3 : coupling WWGammaGamma
18 *
19 L M1 = ( d_(alpha2,mu2)*([k+q](beta1)+p2(beta1))
20       +d_(mu2,beta1)*(-p2(alpha2)+[k+p1](alpha2))
21       +d_(beta1,alpha2)*(-[k+p1](mu2)-[k+q](mu2)) )*
22    ( d_(beta1,beta2)-[k+p1](beta1)*[k+p1](beta2)/Mw^2 )*
23    ( d_(beta2,mu1)*([k+p1](alpha1)+p1(alpha1))
24       +d_(mu1,alpha1)*(-p1(beta2)+k(beta2))
25       +d_(beta2,alpha1)*(-[k+p1](mu1)-k(mu1)) );
26 *
27 L M2 = ( d_(alpha2,mu1)*([k+q](beta1)+p1(beta1))
28       +d_(mu1,beta1)*(-p1(alpha2)+[k+p2](alpha2))
29       +d_(beta1,alpha2)*(-[k+p2](mu1)-[k+q](mu1)) )*
30    ( d_(beta1,beta2)-[k+p2](beta1)*[k+p2](beta2)/Mw^2 )*
31    ( d_(beta2,mu2)*([k+p2](alpha1)+p2(alpha1))
32       +d_(mu2,alpha1)*(-p2(beta2)+k(beta2))
33       +d_(beta2,alpha1)*(-[k+p2](mu2)-k(mu2)) );
34 *
35 L M3 = ( d_(alpha1,mu1)*d_(alpha2,mu2)+d_(alpha1,mu2)*d_(alpha2,mu1)
36      -2*d_(mu1,mu2)*d_(alpha1,alpha2) );
```

In `form`, all variables must be declared before being used. The types of variables can be: Symbol, Index, Vector, Tensor, (C) Function, ... (this is not an exhaustive list).
Note that the variable names can include operators such as + or / provided that the character string is enclosed in square brackets ([]) (see line 3). Line 4 declares indices that are n dimensions (that is, they take values between 1 and n).

Then, we define four local expressions (local means that the scope of these expressions will be the file) starting with L. The expressions `M1`, `M2`, `M3` and `R` correspond to the equations (10.11), (10.12), (10.13) and (10.14) without the coupling terms and without the denominators. The form `k(mu1)` where `k` is a vector and `mu1` an index corresponds to k^{μ_1}, `d_(mu1, mu2)` à $g^{\mu_1 \mu_2}$[1]

Module 2

[1] There is an important subtlety, `form` works in a Euclidean space and not Minskowkien. Most of the time it does not change anything because we do not work with the components of the vectors, `form` transforms into a scalar product and it is up to the user to give it a value. However, when working with γ_5, i factors are missing ....

```
37 *
38 L R = ( d_(alpha1,alpha2)-[k+q](alpha1)*[k+q](alpha2)/Mw^2
39   -k(alpha1)*k(alpha2)/Mw^2+k(alpha1)*[k+q](alpha2)*k.[k+q]/Mw^4 );
40 .sort
41 *
42 * Definition of the different propagators : D0 = k.k - Mw^2,
43 * D1 = [k+p1].[k+p1] - Mw^2,
44 * D2 = [k+p2].[k+p2] - Mw^2, D3 = [k+q].[k+q] - Mw^2
45 * One tries to reconstruct, in the numerator, the different propagators
46 *
47 * First diagram : 1/D0/D1/D3
48 *
49 hide M1,M2,M3,R;
50 L T1 = M1*R;
51 id [k+q] = [k+p1]+p2;
52 id p2.p2 = 0;
53 id k.[k+p1] = D1-k.p1+Mw^2;
54 id p1.[k+p1] = k.p1;
55 id p2.[k+p1] = k.p2+p1.p2;
56 id k.k = D1-2*k.p1+Mw^2;
57 id [k+p1](mu2?) = k(mu2)+p1(mu2);
58 id p1(mu1) = 0;
```

This module concerns the diagram T_1, the strategy is to make `D1` appear in the numerator. On line 44, with the command `hide`, we hide the expressions `M1`, `M2`, `M3` and `R`, that is to say that these expressions are not destroyed, but `form` no longer works with them.

With the command `id`, we replace a quantity with an expression, for example line 46, `id p2.p2 = 0` will replace in all expressions in memory (not hidden) the scalar product `p2.p2` by zero. Note that the replacement rules are only valid for a module. Line 52, we use a wildcard `id [k + p1](mu2?)  = K(mu2) + p1(mu2);` which means whatever the index `mu2`, we replace `[k + p1]` by `k + p1` (attention `mu2` must be declared as `Index`).

On line 56, the command `b D1;` (b for braket) allows you to write the expressions by ordering them according to the powers of `D1`. Line 57, the command `print` writes the active expressions (we can obviously specify to write only certain expressions!).

Module 3

```
59 id p2(mu2) = 0;
60 id [k+p1].[k+p1] = D1 + Mw^2;
61 b D1;
62 print;
63 .sort
64 *
65 * Second diagram : 1/D0/D2/D3
66 *
67 hide T1;
68 L T2 = M2*R;
69 id [k+q] = [k+p2]+p1;
70 id p1.p1 = 0;
71 id k.[k+p2] = D2-k.p2+Mw^2;
72 id p2.[k+p2] = k.p2;
```

```
73 id p1.[k+p2] = k.p1+p1.p2;
74 id k.k = D2-2*k.p2+Mw^2;
75 id [k+p2](mu1?) = k(mu1)+p2(mu1);
76 id p1(mu1) = 0;
```

Same thing for the T_2 diagram, this time we try to make `D2` appear in the numerator.

Module 4

```
77 id p2(mu2) = 0;
78 id [k+p2].[k+p2] = D2 + Mw^2;
79 b D2;
80 print;
81 .sort
82 *
83 * Third diagram : 1/D0/D3
84 *
85 hide T2;
86 L T3 = M3*R;
87 id [k+q] = k+p1+p2;
88 id p1(mu1) = 0;
```

For the diagram T_3, there is no reason to make appear a propagator in the numerator.

Module 5

```
89 id p2(mu2) = 0;
90 id p1.p1 = 0;
91 id p2.p2 = 0;
92 print;
93 .sort
94 *
95 * The coefficients A, B and C are built such that:
96 * Tot(mu1,mu2) = A/p1.p2*p1(mu2)*p2(mu1) + B/p1.p2*p1(mu1)*p2(mu2)
97 * + C*d_(mu1,mu2)
98 *
99 Symbol [1/(n-2)];
100 drop M1,M2,M3,R;
101 hide T3;
102 L Tot = (T1/D1+T2/D2+T3)/D0/D3;
103 * Uncomment this line
104 *L C = 1/2*( d_(mu1,mu2)*Tot-p1(mu2)*p2(mu1)*Tot/p1.p2
105 *          -p1(mu1)*p2(mu2)*Tot/p1.p2 );
106 * and comment the following line for a computation in four dimension
107 L C = [1/(n-2)]*( d_(mu1,mu2)*Tot-p1(mu2)*p2(mu1)*Tot/p1.p2
108          -p1(mu1)*p2(mu2)*Tot/p1.p2 );
109 * end comment
110 L B = p1(mu2)*p2(mu1)*Tot/p1.p2-C;
```

At line 95, the command `drop` permanently deletes the expressions listed.

We construct the coefficients A, B and C as defined by the equations (10.19), (10.20) and

(10.21).

Module 6

```
111 L A = p1(mu1)*p2(mu2)*Tot/p1.p2-C;
112 id p1.p1 = 0;
113 id p2.p2 = 0;
114 id p1(mu1) = 0;
115 id p2(mu2) = 0;
116 b D1,D2,D0,D3;
117 print;
118 .sort
119 drop Tot;
120 id k.k = D0 + Mw^2;
```

Here we use that `form` does not replace the denominator, on line 115 for example, we express `D1` in terms of other propagators to reduce terms of the type $D_1^n/(D_0\, D_3)$.

Module 7

```
121 id k.p1 = (D1-D0)/2;
122 id k.p2 = (D2-D0)/2;
123 id D1 = D3-D2+D0-2*p1.p2;
124 id D0 = D1+D2-D3+2*p1.p2;
125 id D2 = D3-D1+D0-2*p1.p2;
126 b D1,D2,D0,D3;
127 print;
128 .sort
129 *
130 * The integrals over the terms containing one propagator are the same
131 * (it is sufficient to shift k)
132 *
133 CFunction f;
134 id D0^-1*D1^-1*D3^-1 = f(0,1,3);
135 id D0^-1*D2^-1*D3^-1 = f(0,2,3);
136 id D0^-1*D3^-1 = f(0,3);
137 id D1^-1*D3^-1 = f(1,3);
138 id D2^-1*D3^-1 = f(2,3);
139 id D0^-1 = f(0);
140 id D1^-1 = f(1);
141 id D2^-1 = f(2);
```

We introduce a comutting function `f` (defined with `CFunction`). Note that in `form` a function is just an object which obeys certain rules, we don't have to define it as in `C++` for example. Therefore, the number of arguments of the function is not specified. Note also that the order of lines 126-134 matters!

Module 8

```
142 id D3^-1 = f(3);
```

```
143 id f(3) = f(0);
144 id f(2) = f(0);
145 id f(1) = f(0);
146 id f(0,2,3) = f(0,1,3);
147 b f;
148 print;
149 .sort
150 *
151 * Check of the transversality :
152 * the coefficient B plays no role because it is the coefficient
```

Module 9

```
153 * of a term which vanishes when it is contracted with polarisation vectors,
154 * one can check that C = -A, it remains a factor
155 * (d_(mu1,mu2)-p1(mu2)*p2(mu1)/p1.p2)
156 *
157 hide A,B,C;
158 L test = C+A;
159 print;
160 .sort
161 *
162 * J(z) is the integral define in the g+g --> H notes
163 * K = i/(4 \pi)^{n/2} \Gamma(3-n/2)
164 *
165 Symbol [2-n/2],Mh,zw,K;
166 CFunction J,It,ln;
167 unhide A,B,C;
168 drop A,test;
```

Line 159, the command `unhide` activates the following expression list.

Module 10

```
169 id n=2*(2-[2-n/2]);
170 id f(0,1,3) = -K*J(zw)/Mw^2;
171 id f(0,3) = K*(1/[2-n/2]-It(zw));
172 id f(1,3) = K*(1/[2-n/2]-ln(Mw^2));
173 id f(2,3) = K*(1/[2-n/2]-ln(Mw^2));
174 b K,J,[2-n/2];
175 print;
176 .sort
177 *
178 * The divergences drop out and one can take safely n=4
179 * zw = Mw^2/Mh^2
180 *
```

Module 11

```
181 id [2-n/2]=0;
182 id [1/(n-2)]=1/2;
183 id p1.p2 = Mh^2/2;
```

```
184 id Mh^2=Mw^2/zw;
185 b K,J;
186 print;
187 .sort
```

To support the remark (1) of the subsection 10.4.1, we can uncomment lines 5 and 98-99 and comment on lines 6 and 100-101 and run the program again, we will then notice that the result is very different!